Client/Server Tradeoffs for Online Elections

Ivan Damgård and Mads Jurik

Aarhus University, Dept. of Computer Science, **BRICS**[*]

Abstract. We present various trade offs for voting schemes which, compared to known solutions, allow voters to do less work at the expense of more work done by the tallying servers running the election. One such scheme produces ballots of essentially minimal size while keeping the work load on the tally servers on a practical level. Another type of trade off leads to a voting scheme that remains secure, even if an adversary can monitor all client machines used by voters to participate. This comes at the price of introducing an additional party who is trusted to carry out registration of voters correctly.

1 Introduction

Voting schemes is one of the most important examples of advanced cryptographic protocols with immediate potential for practical applications. The most important goals for such protocols are

– Privacy: only the final result is made public, no additional information about votes will leak
– Robustness: the result correcly reflects all submitted and well-formed ballots, even if some voters and/or possibly some of the entities running the election cheat.
– Verifiability: after the election, the result can be verified by anyone.

Other properties may be considered as well, such as receipt-freeness, i.e, voters are not able to prove after the fact that they voted for a particular candidate, thereby discouraging vote-buying or coercing.

Various fundamentally different approaches to voting are known in the literature: one may use blind signatures and anonymous channels[6], where the channels can be implemented using MIX nets (see [2,1] for instance) or based some physical assumption. Another approach is to use several servers to count the votes and have voters verifiably secret share votes among the servers [8,7]. Finally, one may use homomorphic encryption, where a voter simply publishes an encryption of his vote. Encryptions can be combined into an encryption of the result, and finally a number of decryption servers can cooperate to decrypt the result [4], assuming the private key needed for this is secret-shared among them.

[*] Basic Research in Computer Science,
Centre of the Danish National Research Foundation.

Since anonymous channels are quite difficult to implement in practice and verifiable secret sharing requires communication between a voter and all servers, the third method seems the most practical, and this paper deals only with variants of this approach.

In the following, we let L be the number of candidates, M the number of voters, w the number of decryption servers, and k the security parameter for the cryptosystem used. We assume for simplicity that each voter can vote for one candidate. In [4], a solution was given that may be based on any homomorphic threshold encryption scheme if the scheme comes with certain associated efficient protocols. One example of this is El Gamal encryption. The ballot size in this scheme is $O(\log M + b)$ where b is the block size that the encryption scheme is set up to handle. The scheme was designed for the case of $L = 2$, and the generalization to general L given in [4] has complexity exponential in L for the decryption of the final result. Even for $L = 2$, an exhaustive search over all possible election results is required to compute the final result. Therefore, this scheme does not scale well to large elections with many candidates.

In [5,3], solutions were given using a variant of the approach from [4], but based on Paillier's cryptosystem. These are the first solutions that scale reasonably well to large elections, still the most efficient of these protocols produce ballots of size $O((\log L)max(k, L\log M))$. As long as $k > L\log M$, this is logarithmic in L, but for larger values of L and M it becomes linear in L, and each voter has to do $\Omega(\log L)$ exponentiations using a modulus of length $L\log M$ bits. In a real application, one must assume that voters typically have only rather limited computing power available, so that the computation and communication needed for each voter is a rather critical parameter. On the other hand, decryption servers can be expected to be high-end machines connected by high-speed networks.

Thus for a large scale election, it is reasonable to consider the possibility of moving work away from voters at the expense of increased load on the servers. The central issue here is how much we can expect to reduce the size of ballots, since both communication and computational complexity for the voter is directly linked to this parameter. A moments reflection will show that there is no fundamental reason why the size of a ballot should depend on M or be linear in L. Of course, a ballot must be at least $\log L$ bits long, since otherwise we cannot distinguish between the L candidates. Also, it would be unreasonable to expect the encryption to be secure if the size of an encryption (a ballot) did not increase with the security parameter k. Thus a ballot size of $O(k + \log L)$ bits would be essentially optimal. In principle, this is easy to achieve: each voter V_i publishes an encryption of v_i (the id of the candidate he votes for), and the decryption servers use generic multiparty computation [13] to produce securely the result. This is always possible because the encryptions and the decryption key which is shared among the servers together determine the result and could be used to compute it efficiently if they were public. Such a solution, however, would be much too inefficient to have any practical value. It would increase the complexity for the servers by a factor corresponding to at least the size of a Boolean circuit computing the decryption.

In this paper, we present a solution that achieves ballot size $O(k + \log L)$ bits and where each server needs to broadcast $O(ML(k + L \log M))$ bits. Most of this work can be done in a preprocessing phase, and only $O(M(k + L \log M))$ bits need to be sent while the election is running. We assume the random oracle model and that a static adversary corrupts less than w/2 servers and any number of voters. Then the protocol can be proved to be private, robust and verifiable, based on semantic security of Paillier's public key system and the strong RSA assumption. We also present a variant with somewhat larger voter load, where the ballot size is $\log L(k + L)$ bits. This is still less than previous Paillier-based solutions, the communication per server is $O(M \log M(k + L \log M))$ bits. Also here, preprocessing is possible, leading to the same on-line cost as before. This variants can be proved secure in the random oracle model in the same sense as the previous variant, but assuming only semantic security of Paillier's public key system. Both variants can be executed in constant-round. None of the variants are receipt-free as they stand, but under an appropriate physical assumption, they can be made receipt-free using the techniques of [14].

Previous solutions based on the same assumption require each server to read each voters encrypted vote, process this, and broadcast a single piece of data. This amounts to communication that is linear in M, like in the systems we propose here. Thus the extra cost for servers in our solution is that more rounds of interaction are required and that the amount of communication is increased by a factor of L or $\log M$.

The main new technique we use is to have voters work with a cryptosystem with block size $max(k, \log L)$. The servers then securely transform this to encryptions in a related cryptosystem with block size $max(k, L \log M)$, and compute the result using this second system. On top of this, we borrow some techniques from [9].

We note that optimal ballot size can also be achieved using the approach mentioned above based on anonymous channels, where the channels can be implemented using a MIX network. This is because the MIX net hides the origin of a ballot, therefore all ballots can decrypted after mixing and vote counting becomes trivial. For some MIX implementations we get communication complexity for the servers comparable to what we achieve here. However, all known efficient implementations of MIX networks are based on El Gamal encryption, so that the alternative of basing the protocol on Paillier encryption is not available under the MIX approach. Moreover, and perhaps more importantly, it seems to be inherent in the MIX approach that MIX servers do their work sequentially, i.e., each MIX server can only act after the previous one has completed (part of) its work. By contrast, the threshold cryptography approach we use allows servers to complete the protocol in a constant number of rounds. Finally, using a MIX net, it is not clear that one can push most of the server work into a preprocessing phase, as we do here.

The final tradeoff we present is of a completely different type, that relates more to practical security of elections: one of the worst potential weaknesses of electronic voting in practice is that voters are likely to be non-expert computer users, and most likely will use their own machines, home PCs, to cast votes, say over the Internet. Whereas tools such as SSL plus signed applets can be used to give reasonable assurance that the client software used for this is genuine, it is very difficult (some would say impossible) to make sure that the user's machine is not infected by a virus that would monitor key strokes etc., and later transmit these to an adversary who could then easily find out who the voter voted for. By contrast, it seems like a more reasonable assumption that for instance a high-security server placed at some neutral site is not corrupted.

Motivated by this, we propose a solution with the following properties: privacy for the voter is ensured, even if his machine is completely monitored by an adversary, who can follow key strokes, screen image, mouse events, etc. Correctness of the result is ensured, assuming that a particular trusted party, who takes part in registering voters, behaves correctly (cheating will not allow him to break the privacy, however). Whereas this party can in principle be held accountable and can be caught if he cheats, such verification is rather cumbersome. Hence, in practice, this solution trades trust in client machines against some amount of trust in a designated party. We note that a natural candidate for such a player often exists anyway in traditional manual voting schemes, and so in fact no "new" trust is needed - we discuss this in more detail later.

The basic idea of this solution is quite general. It can be combined with our first tradeoff without significant loss of efficiency, but can also be applied to a very simple multicandiate election protocol, that can be based on Paillier encryption or on El-Gamal, and requires the servers to do only L decryptions.

2 The Minimal Vote System

In this section we introduce a scheme in which ballots are of essentially minimal size. This requires that a transformation of the votes are performed by the tally servers to a larger representation of the vote. From the transformed vote the result of the election can be found using the homomorphic properties as usual ([5], [4]).

2.1 Needed Properties

In the reduction of the voter load we need a pair of public-key cryptosystems CS_1 and CS_2 with their respective encryption and decryption functions E_1, E_2, D_1, D_2 . An encryption of m in CS_i under public key pk using random input r will be denoted $E_i(pk, m, r)$, but we will suppress the public keys from the notation as they are kept fixed at all times once generated. We will also often suppress r from the notation for simplicity. N_1 and N_2 will denote the size of the plaintext space for CS_1 and CS_2. The 2 cryptosystems should satisfy:

– **Semantically secure**: Both CS_1 and CS_2 are semantically secure.

- CS_2 **is a threshold system**: The private key in CS_2 can be shared among w decryption servers, such that any minority of servers have no information on the key, whereas any majority of servers can cooperate to decrypt a ciphertext while revealing no information other than the plaintext.
- CS_2 **is homomorphic**: There exists an efficiently computable operation denoted $\otimes$ that when applied to two ciphertexts yield an encryption of the sum of the two plaintexts, that is, we have: $E_2(m_1 \bmod N_2) \otimes E_2(m_2 \bmod N_2) = E_2(m_1 + m_2 \bmod N_2)$. Furthermore, given $\alpha \in Z_{N_2}, E_2(m)$ it is easy to compute an encryption $E_2(\alpha m \bmod N_2)$.
- CS_2 **supports MPC multiplication**: There exists an interactive protocol, denoted MPC multiplication, that the decryption servers can execute on input two encryptions. The protocol produces securely a random encryption containing the product of the corresponding plaintexts, in other words, we can produce $E_2(m_1 m_2 \bmod N_2, r_3)$ from $E_2(m_1 \bmod N_2, r_1)$ and $E_2(m_2 \bmod N_2, r_2)$ without revealing information about m_1 or m_2.
- **Interval proofs**: There exists a zero-knowledge proof of knowledge (that can be made non-interactive in the random oracle model) such that having produced $E_i(m)$, a player can prove in zero-knowledge that m is in some given interval I. For optimal efficiency we will need that the length of the proof corresponds to a constant number of encryptions. For the special case of $I = 0..N_i$ $(i = 1, 2)$, this just amounts to proving that you know the plaintext corresponding to a given ciphertext.
- **Transformable**: There exists a number $B \leq N_1$ such given an encryption $E_1(m, r)$ where it is guaranteed that $m \leq B$, there is an interactive protocol for the decryption severs producing as output $E_2(m, r)$, without revealing any extra information.
- **Random value generation**: The decryption servers can cooperate to generate an encryption $E_2(R)$ where R is a random value unknown to all servers.
- **Vote size**: $L \leq B \leq N_1$ so that votes for different candidates can be distinguished and encryptions be transformed.
- **Election size**: Let $j = \lceil \log_2 M \rceil$. We need that $(2^j)^L < N_2$ to ensure that we do not get a overflow when the final result is computed.
- **Factorization of** N_2: All prime factors of N_2 are super-polynomially large in the security parameter.

We do not want to give the impression this set-up is more general that it really is. We know only one efficient example of systems with the above properties, this example is described below. But we stick to above abstract description to shield the reader from unnecessary details, and to emphasize what the essential properties are that we use.

Example 1. We present a pair of cryptosystems that satisfy the above requirements. This is based on the Damgård-Jurik generalization of Paillier's cryptosystem presented in [5]. A short definition of the basic scheme without going into details about threshold decryption (which can be found in [5]):

DJ (n,s):

- **Public Key**: (n, s), where $n = pq$, p, q primes.
- **Private Key**: d, where $d = 0 \bmod \lambda$ ($\lambda = \mathrm{lcm}(p-1, q-1)$) and $d = 1 \bmod n^s$.
- **Plaintext space**: $\mathcal{Z}_{n^s}$
- **Ciphertext space**: $\mathcal{Z}^*_{n^{s+1}}$
- **Encryption**: $E(m) = (n+1)^m r^{n^s} \bmod n^{s+1}$, where $r \in \mathcal{Z}^*_n$ is random.
- **Decryption**: $L(c^d \bmod n^{s+1})$, where $L(x) = \frac{x-1}{n}$.

Given a n we can choose $CS_1 = DJ(n, s)$ and $CS_2 = DJ(n, s')$ where $s \leq s'$. To satisfy the semantic security condition we need the *decisional composite residuosity assumption* (DCRA), which was introduced by Paillier in [15]:

Conjecture 1. Let A be any probabilistic polynomial time algorithm, and assume A gets n, x as input. Here n has k bits, and is chosen as described above, and x is either random in $Z^*_{n^2}$ or it is a random n'th power in $Z^*_{n^2}$. A outputs a bit b. Let $p_0(A, k)$ be the probability that $b = 1$ if x is random in $Z^*_{n^2}$ and $p_1(A, k)$ the probability that $b = 1$ if x is a random n'th power. Then $\mid p_0(A, k) - p_1(A, k) \mid$ is negligible in k.

Given the conjecture and the transformation shown below, we have all the properties satisfied:

- Semantically secure: Under the DCRA both CS_1 and CS_2 are semantically secure.
- CS_2 Homomorphic: The Damgård-Jurik cryptosystem is homomorphic, where the $\otimes$ operation is multiplication modulo $n^{s'}$. Also we have $E(m)^\alpha \bmod n^{s'+1} = E(\alpha m \bmod n^{s'})$.
- CS_2 supports MPC multiplication: An efficient protocol is shown in [9], requiring each server to broadcast a constant number of encryptions.
- Interval proofs: The proof construction in Appendix A constructs the required proof using communication equivalent to a constant number of encryptions.
- Random value generation: The decryption servers do the following: each server $0 < i \leq w$ chooses at random $R_i \in Z_{N^2}$. The values $E_2(R_i)$ are published followed by zero-knowledge proofs that R_i is known by server i. These proofs can be done using the Multiparty Σ-protocol technique from section 6 of [9] allowing the zero-knowledge proofs to be done concurrently in a non malleable way.
  We then form $E_2(R) = E_2(R_1) \otimes ... \otimes E_2(R_w)$. Thus R is random and unknown to all servers.
- Transformable: An encryption in CS_1 can be transformed to encryption in CS_2 by using the method described below. This method requires that the bound B on the input message satisfies $\log(B) \leq \log(N_1) - k_1 - \log(w) - 2$, where k_1 is a secondary security parameter ($k_1 = 128$ for instance).
- Vote size: we need $L \leq B$ which as mentioned above means $\log(L) \leq \log(N_1) - k_1 - \log(w) - 2$. For most realistic values of k, k_1, L, w this will be satisfied even with $s = 1$, but otherwise s can always be increased.

- Election size: $M^L < N_2 = n^{s'}$ can be satisfied by choosing s' large enough.
- Factorization of N_2: we have $N_2 = n^{s'} = (pq)^{s'}$ and p, q must of course be large to have any security at all.

We now show how to transform the ciphertext $E_1(m)$ from CS_1 to CS_2, where it is known that $0 \leq m \leq B$. Our transformation will work if $\log(B) \leq \log(N_1) - k_1 - \log(w) - 2$.

The crucial observation is that a ciphertext $E_1(m)$ in CS_1 can always be regarded as a ciphertext in CS_2, simply by thinking of it as a number modulo $n^{s'+1}$. It is not hard to see that as a CS_2 encryption, it is an encryption of a number $m' \in Z_{n^{s'}}$ with $m' = m \bmod n^s$. This is not good enough since we want $m = m' \bmod n^{s'}$. All we know is that $m' = m + tn^s \bmod n^{s'}$ for some t we cannot directly compute. To get around this, we mask m with some random bits so that we can find t by decryption, as detailed below.

The masking can be done in 2 ways:

- **Trusted Third Party**: A trusted third party generates a random value R of size $log(B) + k_1$. The 3rd party reveals the value $E_2(R)$.
- **MPC approach**: The servers each generate a value R_i of length $log(B)+k_1$ bits, reveal $E_2(R_i)$ and prove they have done so using an interval proof. This should be done using the Multiparty Σ-protocol technique of [9]. All encryptions with correct proofs are combined using the homomorphic property to get (I is the set of servers which supplied a correct proof):

$$E_2(R) = \Pi_{i \in I} E(R_i) = E(\Sigma_{i \in I} R_i)$$

  This means that R is at most $w2^{log(B)+k_1}$.

Note that the condition on B and R ensures that $m + R < N_1$.

1. We consider the encryption $e = E_1(m)$ as a ciphertext e in CS_2. As noted above, this will be the encryption $E_2(m + tn^s \bmod n^{s'}, r)$ for unknown t and r.
2. Now let $e' = e \otimes E_2(R)$.
3. The servers decrypt e' to get a message $m + R + tn^s \bmod n^{s'}$. Since we have $m + R < N_1$, we can find $m + R$ and t just by integer division. And if at least one server has chosen its R_i at random, information on m will be statistically hidden from the servers, since R is at least k_1 bits longer than m.
4. We now set $e'' = e \otimes E_2(-tn^s, 1)$. Due to the homomorphic properties this is equal to $E_2(m)$.

2.2 Preparation

The preparation phase requires the generation of the 2 cryptosystems with key distribution for threshold decryption in CS_2. We also need a publicly known polynomial of degree $L - 1$ which satisfies the equation:

$$f(i) = M^i \bmod N_2 \qquad \forall i : 0 \leq i < L$$

By assumption N_2 has only very large prime factors. Hence any difference of form $i - j$ where $0 \leq i, j < L$ is invertible modulo N_2 and this is sufficient to ensure that f can be constructed using standard Lagrange interpolation.

The next and last part of the preparation has to be done once for each election. For each voter, the severs generate a random encryption $E_2(R)$ as described earlier. Then we execute $L-2$ MPC multiplications to get encryptions $E_2(R^j)$ for $j = 1, ..., L-1$.

2.3 Voting

The voter generates a vote for candidate i by making an encryption $E_1(i)$ and an interval proof that it is the encryption of a value in the interval $\{0, ..., L-1\}$.

2.4 Transformation

When the servers receive the vote as a ciphertext in CS_1 they have to transform it into a corresponding vote in CS_2, that can be added together to give a meaningful result. This is done by transforming $E_1(i)$ to $E_2(M^i)$. This has to be done for each vote and can be done in the following way:

1. We transform $E_1(i)$ into $E_2(i)$.
2. The servers decrypt $E_2(i) \otimes E_2(R)$ to get $z = i + R$. It follows that $i^j = (z - R)^j$, and this can be rewritten using the standard binomial expansion. The result is that $i^j = \alpha_0 + \alpha_1 R + ... + \alpha_j R^j$ for publicly known values $\alpha_0, ..., \alpha_j$. Hence encryptions $E_2(i^j)$ can be computed without interaction from the encryptions $E_2(R^j)$ from the preparation phase, using the homomorphic property. From these encryptions, we can, using the polynomial f computed in the preparation, construct an encryption $E_2(f(i))$, still with no further interaction. The result of this satisfies $E_2(f(i) \bmod N_2, r) = E_2(M^i \bmod N_2, r)$

2.5 Calculating Result

Now we can combine all the transformed votes using the homomorphic property of CS_2 and decrypt the result. This will give a value of the form:

$$\sum v_i M^i \qquad \forall i : 0 \leq v_i < M$$

Since M is the number of voters an overflow of v_i mod M cannot have occurred and since $M^L < N_2$ we get that the number of votes on the i'th candidate will be v_i.

2.6 Complexity

From the voters point of view the computational (modular multiplications) and communicational complexity (bits) of this protocol will be $O(log(L) + k)$. This is within a constant of the smallest possible.

The decryption servers' work depends on the cryptosystems used, and can only really be compared in the number of usages of the primitives: transformations (from CS_1 to CS_2), decryptions, MPC multiplications, and random value generations.

In the preparation, we generate M random values and do $M(L-2)$ MPC Multiplications. During election we do M transformations from CS_1 to CS_2 and M decryptions, plus 1 to get the result. To calculate the powers of R in the preprocessing, $O(L)$ rounds of communication are needed. Constant round solutions can also be devised using techniques from [16], but the total communication will be larger. The protocol for the election itself is constant round.

3 An Alternative System

Here we look at an alternative scheme that requires more work for the voter, but the work required by the tallying servers can be reduced compared to the previous scheme for some parameter values.

3.1 Needed Properties

In this trade off scheme we also need a pair of cryptosystems CS_1 and CS_2 with properties as described earlier, except for two changes:

- **Zero-knowledge proofs**: Interval proofs are not needed for this scheme. Instead we need that a player can generate an encryption $E_1(v)$ and prove in zero-knowledge that $v \in \{2^0, ..., 2^{L-1}\}$. For the example of Paillier based encryption, a protocol for this purpose is given in [5].
- **Vote Size**: In this scheme, we need $2^L \leq B \leq N_1$ instead of $\log_2 L \leq B$.

3.2 Preparation

The cryptosystems must be set up as for the previous scheme.

In preparation of each election a pair of values have to be generated for each voter (recall that we defined j to be minimal, such that $2^j > M$):

- An encryption of some random R: $E_2(R \bmod N_2)$.
- The inverse of R raised to the j'th power: $E_2(R^{-j} \bmod N_2)$.

These values are generated before the election so that the result of the election is more efficiently computed when the votes start to arrive. The values can be generated with one of these 2 methods:

- **Trusted third party**: The trusted third party generates the 2 encryptions.
- $O(\log(j))$ **MPC multiplications**: The servers cooperate on generating a random encryption $E_2(R)$. Using the inversion method from [16] the value

$E_2(R^{-1})$ is generated [1]. Then the servers use the MPC multiplication $O(log(j))$ times to get $E_2(R^{-j})$.

3.3 Voting

The voter generates a vote for candidate i by setting $v = 2^i$, making $E_1(v)$ and a proof that it is the encryption of a message from the set $\{2^0, ..., 2^{L-1}\}$.

3.4 Transformation

The goal of the transformation is to compute $E_2((v)^j)$ from $E_1(v)$ and can be done as follows:

1. The encryption of the vote v is transformed to $e = E_2(v)$ in CS_2
2. The servers perform a MPC multiplication of $e = E_2(v)$ and $E_2(R)$ to get $e' = E_2(vR \bmod N_2)$
3. The servers decrypt e' to get $vR \bmod N_2$ which reveals no information of v since R is chosen at random (note that by assumption on N_2, both v and R are prime to N_2 except with negligible probability).
4. The servers raises vR to the j'th power in public and make an encryption of this, $e'' = E_2((vR)^j \bmod N_2, 1)$ (we use a default value of 1 for the random input to encryption, no randomness is needed here).
5. The servers make a MPC multiplication of e'' and $E_2(R^{-j} \bmod N_2)$ to get the transformed encryption of the vote
$$E_2(v^j \bmod N_2, r) = E_2((2^j)^i \bmod N_2, r).$$

3.5 Calculating Result

To calculate the result the transformed votes are combined using the homomorphic property of CS_2, and the resulting ciphertext is decrypted. The plaintext from the decryption will have the form:

$$\sum v_i(2^j)^i \qquad \forall i : 0 \leq v_i < 2^j$$

Since $2^j > M$, where M is the number of voters an overflow cannot have occurred for a single candidate and the whole election cannot have caused a overflow since $(2^j)^L < N_2$. The number of votes on the i'th candidate is v_i.

[1] This is done by generating another encryption of a random value R' in the same way as the first. Then compute the MPC multiplication of the 2 and decrypt it to get $RR' \bmod N_2$. This is inverted and encrypted again. Then this is MPC multiplied with $E_2(R')$ again to get $E_2((RR')^{-1}R' \bmod N_2) = E_2(R^{-1} \bmod N_2)$

3.6 Complexity

The communication needed from the voter is now $O(L+k)$, plus the size of the proof of correctness for the encryption (which in the Damgård-Jurik scheme will have size $O(log(L))$ encryptions using the techniques from [5]).

If a trusted third party is used then there is no precomputation for the tally servers, but otherwise they have to generate the pair of values: to generate the inverses we need 1 random value generation, 2 MPC multiplications and 1 decryption, and for calculating the j'th power we need at most $2\log_2(M)$ multiplications which means that we use a total of $M(\log_2(M)+2)$ MPC multiplications and M decryptions and random values.

For the election itself, the number of transformations we need from CS_1 to CS_2 is M. In the protocol we use a decryption when raising each vote to the j'th power, so we need $M+1$ decryptions. And finally we need a total of $2M$ MPC multiplications.

The preparation can be done in $O(log(M))$ rounds, while the protocol after preparation is constant round.

In comparison with the first scheme, we see that the voters do more work here, and the complexity of the election after preparation is comparable, but slightly lower in the first scheme. The main difference is that the complexity of the preparation is $O(ML)$ MPC multiplications in the first scheme and $O(M\log M)$ in the second. Another difference is that the first scheme requires ML encryptions to be stored between preparation and election, while the second scheme requires only $2M$ encryptions.

Thus the second scheme may have an advantage if $\log M$ is less than L. Even for large scale elections, this may well happen, since even if $10^6 \le M \le 10^9$ $\log M$ is only between 20 and 30.

4 Protecting Clients against Hackers

How can a voter be protected against a curious person that has full access to his computer during an election? In this section we look at a way to trade trust in the security of the client computer against trust in a third party. We first describe the basic idea on a high level and then give two ways to implement the idea.

We assume that we have a trusted party T (we discuss later in which sense he has to be trusted). T will for each voter choose a random permutation π permuting the set $0,1,...,L-1$. He then privately (and possibly by non-electronic means) sends a list containing, for each candidate number i, the candidate's name and $\pi(i)$. When using his own (or any) client machine to cast his vote, the voter decides on a candidate - say candidate number i, finds his name on the list, and tells the client software that he votes for candidate $\pi(i)$. The client software could simply present a list of numbers from 0 to L_1 to choose from, without any corresponding names. The client software sends an encryption of $\pi(i)$ to the tally servers.

At the same time as π is generated, T also sends to the tally servers an encryption of π. Using this encryption, the servers can transform the encryption of $\pi(i)$ into an encryption of i, and the election result can then computed using the homomorphic properties as usual.

As for security of this, consider first correctness: as we have described the system, we clearly have to trust that T encrypts for the servers the correct permutation for each voter. If not, the result will be incorrect. Note, however, that T cannot decrypt the encryption of $\pi(i)$ sent from the client machine, so it cannot manipulate the permutation and be certain to favor a particular candidate. If T was suspected of foul play against a particular voter, the information held by the voter could be verified against the encryption of π, and then cheating would always be caught. But since this is a rather cumbersome procedure, it is unlikely to happen very often, and so some amount of trust has to be invested in T.

As for privacy, clearly an attacker monitoring the client machine gets no information on who the voter voted for, by the random choice of π. Furthermore, even if T pools its information with a minority of the severs, they cannot break the privacy of the voter unless they break the encryption. A breach of privacy would require both that T is corrupt and that it participates in an attack where client machines are infected.

In practice, who might play the role of T? As an example, in many countries, there is an authority which, prior to elections and referendums, sends by private paper mail a card to every eligible voter, and this card must be used when casting a vote. Such an authority could naturally play the role of T, and simply print the information about π on the card sent out. In other countries voters must contact a government office to get registered, in this case the permutation could be generated on the fly and the information handed directly to the voter.

For the first implementation of this idea we use a cryptosystem CS with encryption and decryption functions E, D and plaintext space of size N.

4.1 Needed Properties

Here we need less assumptions because we do not need to transform the votes between different cryptosystems.

- CS **is homomorphic**: as defined earlier
- CS **supports MPC multiplication**: as defined earlier
- **Zero-Knowledge proofs**: we need that a player can generate an encryption $E(v)$ and prove in zero-knowledge that $v \in \{M^0, ..., M^{L-1}\}$. For the example of Paillier based encryption, a protocol for this purpose is given in [5].
- **Election size**: To ensure that the final result is correct we need $M^L < N$.
- **Factorization of** N: We assume that N has only very large prime factors so that factoring N is infeasible.

4.2 Preparation

T picks a random permutation π for each voter and gives the $\pi(i)$ values to the voter as described above. Then T generates a polynomial of degree $L-1$ for each of the voters with the property that

$$f(M^i) = M^{\pi^{-1}(i)} \bmod N \quad \forall i : 0 \leq i < L$$

If doing this by Lagrange interpolation fails, this can only be because some number less than N was found to be non-invertible modulo N, which implies N can be factored. Since this was assumed infeasible, the construction fails with negligible probability. The L coefficients of the polynomial are then encrypted to produce encryptions $c_0, ..., c_{L-1}$ and these are given to the tallying servers.

The tally servers for each voter generates a random encryption $E(R)$ and compute encryptions of the powers $E(R^2), ..., E(R^{L_1})$.

4.3 Voting

To make a vote for candidate i the voter gives $\pi(i)$ to the client machine who makes an encryption $E(M^{\pi(i)})$ and appends a zero-knowledge that one of $M^0, ..., M^{L-1}$ was encrypted.

4.4 Transformation

We need to transform the vote $E(M^{\pi(i)})$. First we use the encryptions of powers of R from the preparation to compute encryptions $E(M^{2\pi(i)}), ..., E(M^{(L-1)\pi(i)})$. This is done the same way as in the minimal vote scheme and requires only one decryption and local computation. From this and $c_0, ..., c_{L-1}$, the servers can clearly use the homomorphic property and $O(L)$ MPC multiplication to make an encryption $E(f(M^{\pi(i)}))$. If T participates, it can be done much more efficiently: since T knows the coefficients of f, he can produce $E(f(M^{\pi(i)}))$ from $E((M^{2\pi(i)}), ..., E(M^{(L-1)\pi(i)})$ by only local computation, and prove in zero-knowledge to the servers that this was correctly done. The proof is straightforward to construct using techniques from [9][2].

We then have

$$E(f(M^{\pi(i)})) = E(M^{\pi^{-1}(\pi(i))}) = E(M^i)$$

which is what we wanted.

4.5 Combination

The result can then be found using the homomorphic addition of the transformed votes to get a number of the form:

$$\sum v_i M^i \qquad \forall i : 0 \leq v_i < M$$

since M is the number of voters and $M^L < N$ an overflow cannot have occurred and the number of votes on the i'th candidate will be v_i.

[2] In [9], a zero-knowledge protocol was given by which a player can prove that a committed constant was correctly "multiplied into" a given encryption, and this is exactly what we need here

4.6 Complexity

Since we don't have any reduction in the size of the cryptosystem the voters the communication and computational complexities are $O(L \log M + k)$ plus the size of the proof that the vote has the right form.

For the tallying servers the complexity of both preparation and election is comparable to the minimal scheme in case T participates also in the election. Otherwise we will need $O(ML)$ MPC multiplications during the election itself.

4.7 Combination with Minimal Votes

Since the scheme we just presented is similar to the minimal vote scheme it is straightforward to combine the two. This only adds the cost of transforming the vote from CS_1 to CS_2. The polynomial must now have the form

$$f(i) = M^{\pi^{-1}(i)} \bmod N$$

and the voter send an encryption of form $E_1(\pi(i))$ as his encrypted vote.

4.8 An Alternative Implementation

A very simple way to implement multicandidate elections from homomorphic encryption is as follows: the voter produces encryptions $e_1, ..., e_L$, where $e_i = E(1)$ if he votes for candidate i and all other encryptions contain 0. He proves in zero-knowledge that each e_j encrypts 0 or 1, and opens $e_1 \otimes ... \otimes e_L$ to reveal 1, in order to prove he voted for one candidate. The tally servers can then combine all 0/1 votes for each candidate separately using the homomorphic property and decrypt. This method places a quite large workload on voters, but on the other hand it can be based on El-Gamal encryption as well as on Paillier, and it is the only known way in which elections with large L can be efficiently based on El-Gamal encryption (the method from[4] is exponential in L).

It is straightforward to apply the client protection method to this voting scheme: the trusted party T generates and communicates a permutation to each voter as described above. Then to encrypt a permutation π for the tally servers, T will generate an $L \times L$ permutation matrix M_π representing π^{-1} in the standard way and publish encryptions of each entry in M_π. We let $E(M_\pi)$ denote this (ordered) set of encryptions. The voter will now send a set of encryptions where $e_1, ..., e_L$, where $e_{\pi(i)} = E(1)$. Since T knows the entries of M_π, he can by only local computations and using the homomorphic property produce random encryptions $e'_1, ..., e'_L$ such that $e_i = E(1)$ and all the others contain 0. This is done by applying M_π to the vector of encryptions and multiplying by random encryptions of 0. Since $E(M_\pi)$ was made public, he can then prove in zero-knowledge to the servers that this was correctly done using techniques from [9]. Finally the computation of the final result can be completed as above.

References

1. Abe: *Universally verifiable MIX net with verification work independent of the number of MIX centers*; proceedings of EuroCrypt 98, Springer Verlag LNCS.
2. Ohkubo and Abe: *A Length-Invariant Hybrid Mix* Proceedings of Asiacrypt 00, Springer Verlag LNCS.
3. Baudron, Fouque, Pointcheval, Poupard and Stern: *Practical Multi-Candidate Election Scheme*, manuscript, May 2000.
4. R.Cramer, R.Gennaro, B.Schoenmakers: *A Secure and Optimally Efficient Multi-Authority Election Scheme*, Proceedings of EuroCrypt 97, Springer Verlag LNCS series, pp. 103-118.
5. Damgård and Jurik: *A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System*, Proc. of Public Key Cryptography 2001, Springer Verlag LNCS series.
6. A. Fujioka, T. Okamoto & K. Otha: *A practical secret voting scheme for large scale elections.*, Advances in Cryptology - Auscrypt '92, pp. 244-251.
7. B. Schoenmakers: *A simple publicly verifiable secret sharing scheme and its application to electronic voting*, Advances in Cryptology - Crypto '99, vol. 1666 of LNCS, pp. 148-164.
8. R. Cramer, M. Franklin, B. Schoenmakers & M. Yung: *Multi-authority secret ballot elections with linear work*, Advances in Cryptology - Eurocrypt '96, vol. 1070 of LNCS, pp. 72-83.
9. R. Cramer, I. Damgård and J. Nielsen:*Multiparty Computation from Threshold Homomorphic Encryption*, Proceedings of EuroCrypt 2001, Springer Verlag LNCS series 2045, pp.280-300.
10. Boudot: *Efficient Proof that a Comitted Number Lies in an Interval*, Proc. of EuroCrypt 2000, Springer Verlag LNCS series 1807.
11. Damgård and Fujisaki: *An Integer Commitment Scheme based on Groups with Hidden Order*, Manuscript, 2001, available from the Eprint archive.
12. Fujisaki and Okamoto: *Statistical Zero-Knowledg Protocols to prove Modular Polynomial Relations*, proc. of Crypto 97, Springer Verlag LNCS series 1294.
13. Oded Goldreich, Silvio Micali, and Avi Wigderson: *How to play any mental game or a completeness theorem for protocols with honest majority*, in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, 25–27 May 1987.
14. M.Hirt and K.Sako: *Efficient Receipt-Free Voting based on Homomorphic Encryption*, Proceedings of EuroCrypt 2000, Springer Verlag LNCS series, pp. 539-556.
15. P.Pallier: *Public-Key Cryptosystems based on Composite Degree Residue Classes*, Proceedings of EuroCrypt 99, Springer Verlag LNCS series, pp. 223-238.
16. J. Bar-Ilan, and D. Beaver: *Non-Cryptographic Fault-Tolerant Computing in a Constant Number of Rounds*, Proceedings of the ACM Symposium on Principles of Distributed Computation, 1989, pp. 201-209.

A Interval Proofs for Paillier Encryptions

Given a Paillier encryption $E(m, r)$ (computed modulo N^2), we sketch here an efficient method to prove in zero-knowledge that m is in some given interval I. In [5] a protocol for this purpose is given. However, there one needs to supply an encryption of every bit in the binary expansion of m. We want a more efficient method, where only a constant number of encryptions need to be sent. In the following, *opening an encryption* $E(m, r)$ means revealing m, r. However, for simplicity, we will suppress r in the notation in most cases.

In [10] Baudot gives an efficient method for proving that a committed number lies in a given interval. The protocol requires sending only a constant number of commitments and is zero-knowledge in the random oracle model that we also use here. It assumes that the number has been committed to using a commitment scheme with some specific properties. The scheme proposed by Fujisaki and Okamoto [12] will suffice, assuming the strong RSA assumption. See [10] for a short description of the commitment scheme and associated protocols. It should be noted that there are some technical problems with the proof of soundness for the associated protocols given in [12], but these problems have recently been fixed in [11]. The modulus N used for Paillier encryption can also serve as part of the public key for the commitment scheme in Baudot's protocol. In addition, we need two elements $g, h \in Z_N^*$ of large order, such that g is in the group generated by h. The prover must not know the discrete logarithm of g base h or vice versa. We assume that g, h are generated as part of the procedure that sets up N and shares the private key among the decryption servers. A commitment to m in this scheme using random input r is $Com(m, r) = g^m h^r \bmod N$. We will often just write $Com(m)$ for simplicity.

Now, the basic idea is the following: given $E(m)$, the prover provides a commitment $Com(m)$, proves that the commitment contains the same number as the encryption, and then uses Baudot's protocol to prove that $m \in I$. The only missing link here is how to show that the same number m is contained in encryption and commitment. This can be done as follows:

1. Let T be the maximal bit-length of m (based on the interval I). The prover chooses at random u, an integer of length $m + 2k_1$ where k_1 is the secondary security parameter. He sends $a = E(u), b = Com(u)$ to the verifier.
2. The verifier chooses a k-bit challenge e.
3. The prover opens the encryption $a \cdot E(m)^e \bmod n^2$ and the commitment $b \cdot Com(m)^e \bmod N$, to reveal in both cases the number $z = u + em$. The verifier checks that the openings were correct.

This protocol can be made non-interactive in the standard way using a hash function and the Fiat-Shamir heuristic. It is then also statistically zero-knowledge in the random oracle model.

What we have done is to combine two already known protocols for proving knowledge of the contents of an encryption, respectively a commitment. Then, when we prove soundness of this protocol using a standard rewinding argument, the fact that we use the same challenge e and the same response z is both cases will ensure the prover must know one single value that is inside both the encryption and the commitment. Indeed, if the prover for given a, b could answer two different challenges e, e' by numbers z, z', then this common value would be $(z - z')/(e - e')$. The strong RSA assumption is used here, to show that $e - e'$ must divide $z - z'$, except with negligible probability. Details are deferred to the final version of the paper.